RELATÓRIO FINAL - MODELAGEM DE SOFTWARE
Sistema de Gerenciamento Hospitalar (SGH)

Grupo:

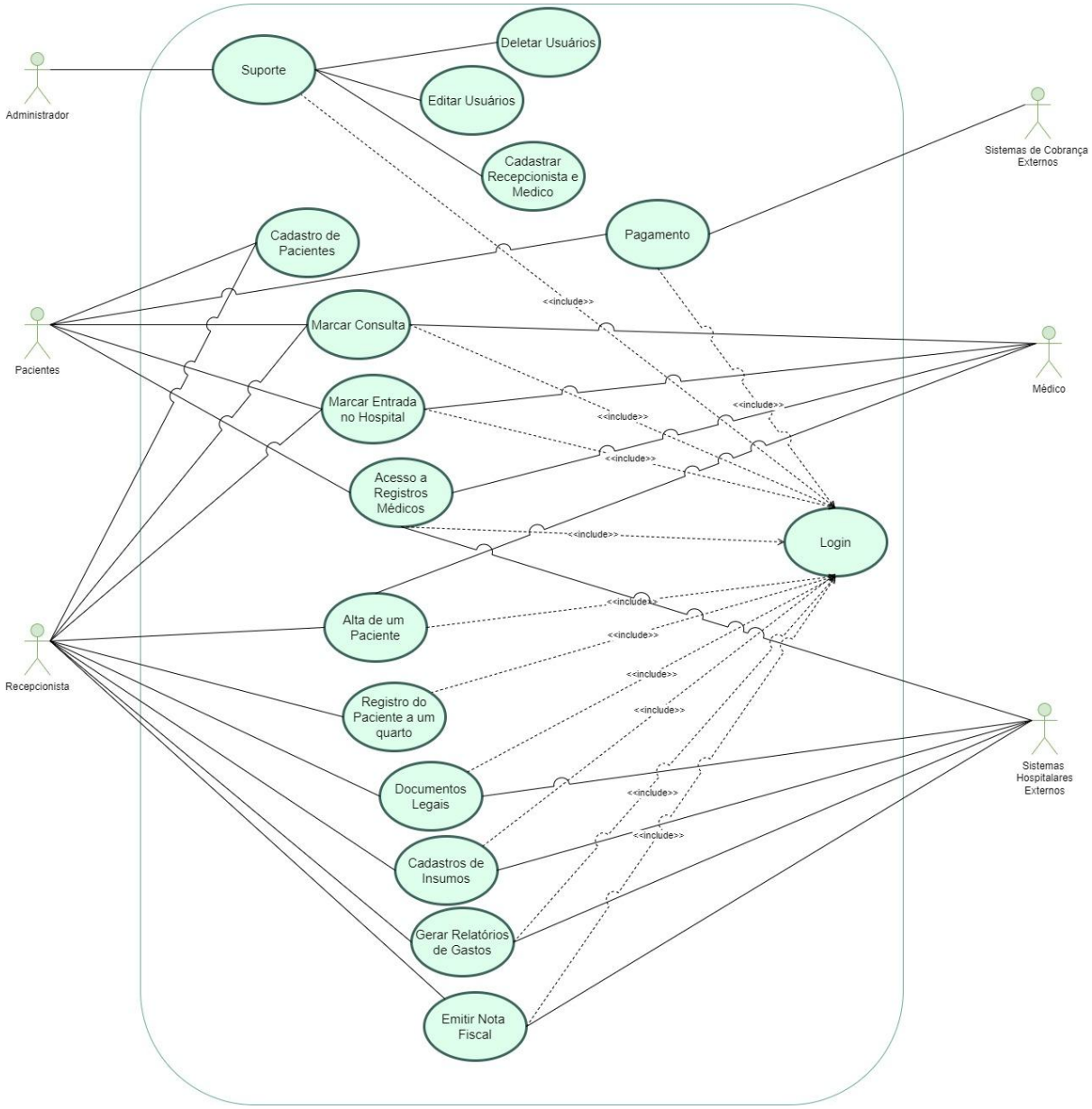
João Vitor Afonso Pereira - 11911BCC037
Maxley Soares da Costa - 11911BCC038

Breve Introdução ao Produto (Link - descrição completa)

O Sistema de gerenciamento hospitalar (SGH) é um sistema web desenvolvido para auxiliar na gestão de pacientes e funcionários de hospitais públicos e particulares. Entre suas funcionalidades estão: cadastro de pacientes, funcionários e médicos do hospital. Também é possível fazer o gerenciamento de consultas médicas, insumos hospitalares, leitos e a partir disso fazer relatórios para gerenciar custos do hospital. O sistema também dá total autonomia ao paciente de agendar suas consultas com o médico de sua preferência. Para uma descrição mais detalhada, acesse o link acima

Use Case (Link - images)

Use Case Name	Emitir Nota Fiscal
Requisito Funcional	14
Pré-condições	Usuário pré-cadastrado, sendo um dos atores primários
Requisição Bem sucedida	Permite que os sistema gere notas fiscais eletrônicas padrão.
Requisição falha	Usuário não logado ou acesso não permitido.
Ator primário	Recepcionista, Sistemas Hospitalares Externos
Gatilho	O recepcionista ou sistema externo em questão solicita a criação de uma nota fiscal em relação a algum pagamento ou compra.
Main Flow	1. O recepcionista ou sistema externo solicita uma nota fiscal no sistema. 2. O sistema verifica a solicitação e o custo relacionado e gera a nota. 3. O sistema gera uma nota eletrônica.
Extensions	1. Não ha o custo selecionado para gerar a nota. 2. O sistema não reconhece o usuário em questão.


Administrador
Suporte
Deletar Usuários
Editar Usuários
Cadastrar Recepcionista e Medico
Sistemas de Cobrança Externos
Pagamento
Cadastro de Pacientes
Pacientes
Marcar Consulta
Médico
Marcar Entrada no Hospital
Acesso a Registros Médicos
Login
Alta de um Paciente
Recepcionista
Registro do Paciente a um quarto
Sistemas Hospitalares Externos
Documentos Legais
Cadastros de Insumos
Gerar Relatórios de Gastos
Emitir Nota Fiscal
<<include>>

Arquitetura do Sistema (Link - images)

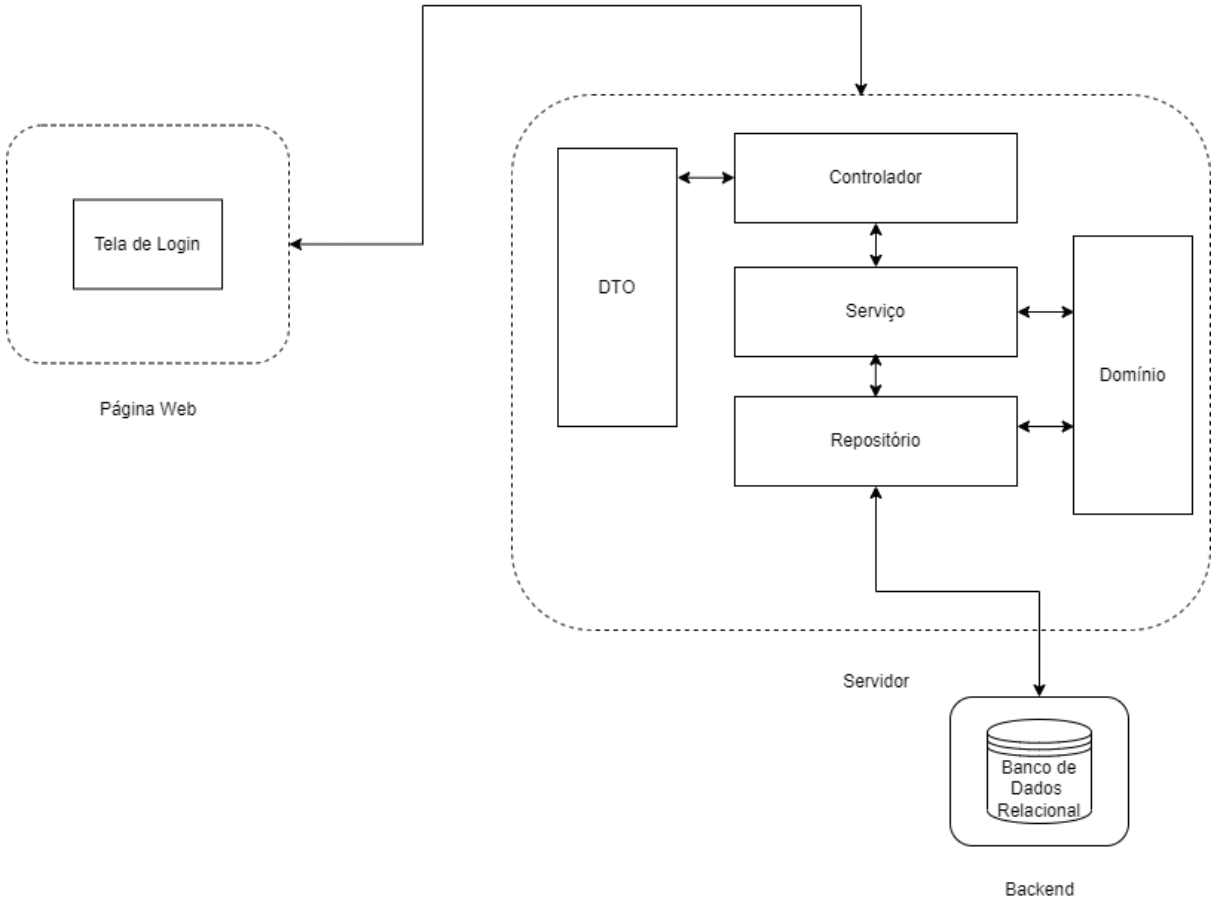


Figura 1: Arquitetura Simplificada.

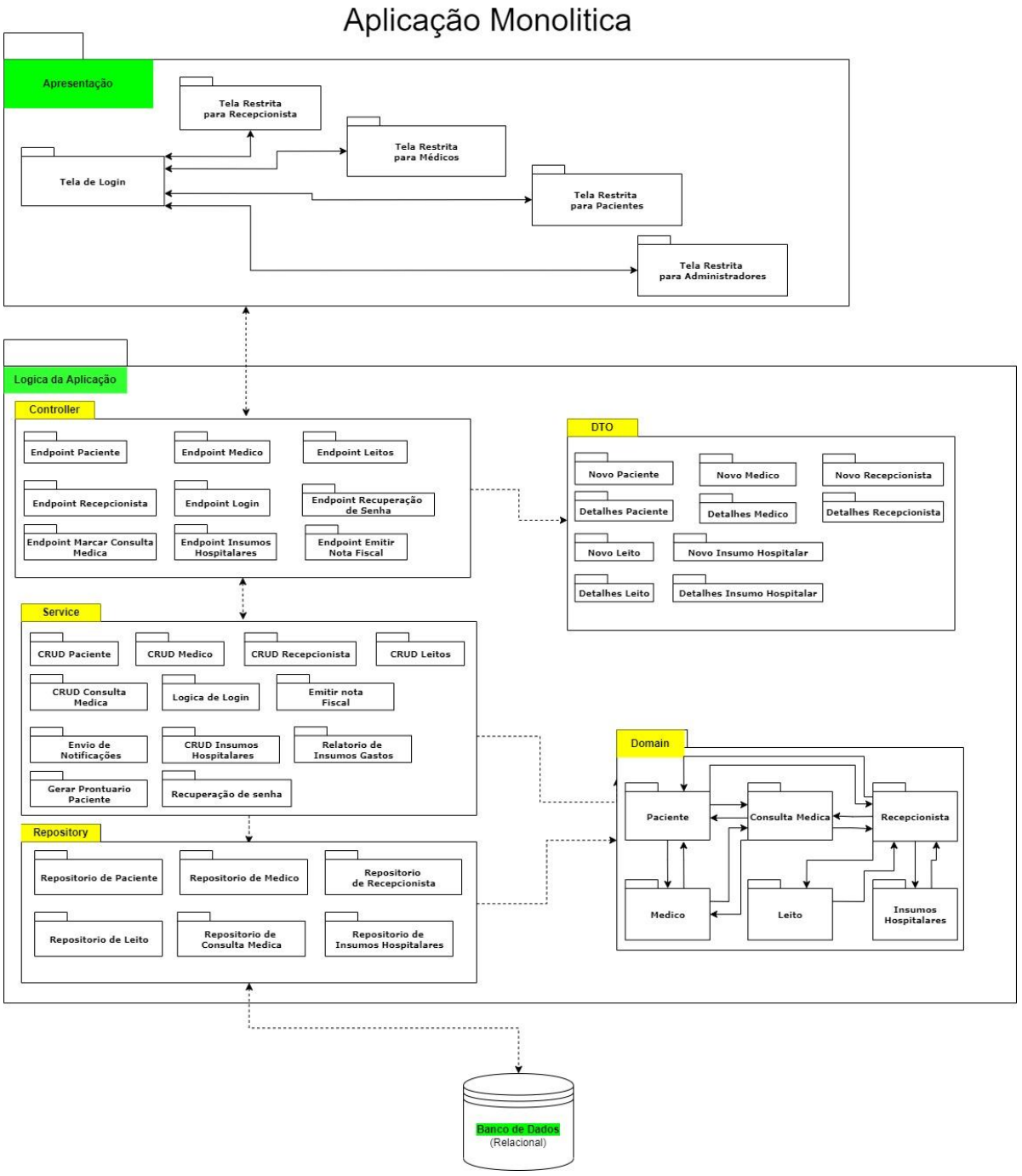


Figura 2: Arquitetura detalhada.

Porque pretendemos adotar MVC

O padrão MVC possui um design modular que separa a representação da informação da interação do cliente. Esse padrão é separado em 3 camadas Model, View e Controller o nos dá muita praticidade tanto na hora de desenvolver novas funcionalidades quanto na hora de dar manutenção nas funcionalidades existentes. Outra característica do padrão que foi levada em consideração é sua flexibilidade para criar protótipos, permitindo que o design do produto e seu código possa andar de maneira simultânea.

Porque pretendemos criar um monolito

Sabemos que no momento atual várias aplicações estão adotando a arquitetura de microsserviços e outras muitas estão migrando do monolito para microsserviços. Mas por que decidimos criar um monolito então? Bom, muitas pessoas possuem uma ideia errada sobre uma aplicação monolítica. Esse tipo de sistema é muito mais simples de ser construído e de dar manutenção uma vez que não precisamos nos preocupar com coisas como integração entre sistemas internos, tracing, etc. Mas e quanto à performance superior que um sistema de microsserviços promete? Antes de pensarmos em ir para uma arquitetura de microsserviços por conta de performance podemos explorar uma gama de melhorias que podem ser feitas no monolito, como escalabilidade vertical (aumentar recursos de hardware) ou horizontal (aumentar a quantidade de máquinas rodando o serviço), tunning do banco de dados para aumentar a performance de leitura e escrita, etc.

Então quando devemos pensar em uma solução utilizando microsserviços? A utilização de microsserviços é recomendada para escalar time mais do que escalar a aplicação. Sendo assim utilizariamos uma solução em microsserviços quando o time tivesse uma quantidade considerável de desenvolvedores de tal maneira que houvesse um gargalo no processo de entrega de novas funcionalidades.

Testes

Para testar nossa aplicação vamos utilizar o processo TDD o qual os testes seriam desenvolvidos antes da codificação do projeto em si, assim inicialmente um teste criado para uma funcionalidade do projeto falharia. A partir dessa falha haveria a codificação da funcionalidade em si com o objetivo de fazer o teste passar. Após o sucesso do teste inicialmente criado, o código do projeto será refatorado para o mesmo estar de conformes das boas práticas do Desenvolvimento de Software, para tornar o mesmo mais limpo e menos acoplado. Esse processo é chamado de red green refactor.

Com o intuito de manter a confiabilidade da aplicação seriam feitos testes unitários para cada funcionalidade do sistema, assim ao fim do desenvolvimento utilizando TDD o projeto estará devidamente testado e refatorado, tornando o desenvolvimento dinâmico e mais confiável para eventuais mudanças no código, seja para melhorá-lo ou alterar regras de negócio.

Também serão realizados testes de integração com o banco de dados para verificar se os dados estão sendo persistidos corretamente e averiguar se as eventuais alterações estão ocorrendo de maneira correta. Como nosso sistema prevê uso de API 's externas, também vamos fazer testes de integração com as mesmas para nos certificar de que os dados que são enviados ainda são os mesmos que nos interessa na aplicação e se estão sendo enviados de maneira que esperamos.

Como ferramenta de testes vamos utilizar o JUNIT juntamente com o Mockito que já possuem diversas ferramentas que vão facilitar no momento de implementação.